Orchestrator API

1. Introduction

The Orchestrator API is part of the Workflow Manager (WM). Globaly, the WM consists of a Session Manager (SM), an Orchestrator and a set of Slurm plugins to control the compute and data nodes.

These components may have common configuration. For the sake of simplicity, for the time being we propose to identify these configurations and to provide them in a common file store on a shared FS.

(we need to discuss what is necessary to be shared in order to avoid or at least minimise duplication of information)

```
#
# Workflow Manager configuration file.
#

# S3 LTStorage server address
#***** required by SM and Orchestrator
# Examples
# s3_server = <S3 hostname |S3 ip address>

# Keycloak server address
#***** required by SM and Orchestrator
# Examples
# keycloak_server = <keycloak hostname | keycloak ip address>

# HSM storage levels names and operations
#***** required by SM
# Examples
# target_tier_list = NVMe, SSD, HDD, tape
# datamover_operation_list = copy, move, release

# MSA layout (compute partitions, data nodes (inventory, capabilities & connections))
#***** required by Orchestrator (the location list is used by the SM to validate the WDF)
# Examples
# Location=<name> ComputeNodes=<nodeset> Datanodes=<nodeset> NeighborLocation=<location list>

# Ephemeral Service specific properties

# Smart Burst Buffer
# Flavors: A flavor defines the size of the memory, the number of cores and the storage space
# that are used by the Smart Burst Buffer when a job requests such a flavor.
# MemorySize specifies the size of the memory of the flavor while StorageSize
# specifies its storage space. They can be set using your preferred unit (e.g. GB).
# Without unit, the size is expressed in bytes.
# Cores specifies the number of cores of the flavor.
# ***** required by RM of Orchestrator (the flavor list is used by the SM to validate the WDF)
# Examples
# Flavor=small MemorySize=5GB Cores=2
# Flavor=medium MemorySize=10GB Cores=4 StorageSize=500GB
# Flavor=large MemorySize=50GB Cores=16 StorageSize=2TB
```

1.1. Workflow Manager Overview

CLI iosea-wf	**Session Manager Endpoints** **(called by CLI)**	**Orchestrator Endpoints** **(called by Session Manager, BB slurm plugin or HSM)**	
		Resource Manager (RM)	**Others endpoints**
start	POST:session/startup • start the session *orchestrator*:ephemeralservice/reserve	POST: server/allocation (only v1) In the second version of the RM API the synchronous server /allocation endpoint is replaced by two asynchronous endpoints: POST: ephemeralservice/reserve (v2) • reserve resources for a service GET: server/allocation/{service_name} (v2) • return information on allocated resources for the *service_name*	

stop	POST: session/stop/{session_name} • check status of all session steps *orchestrator*:step/status/{step_id} • stop all session service *orchestrator*:ephemeralservice /release/{service_name} POST: session/forcedstop/{session_name} • check or abort all session steps *orchestrator*:step/abort/{step_id} • stop all session service *orchestrator*:ephemeralservice /release/{service_name}	DELETE: server/allocation/{service_name}(v2) • remove resource reservation	DELETE: ephemeralservice/release/{service_name} • stop the *service_name* ephemeral service
run	POST: step/startup/{step_name} • launch a step instance *orchestrator*:step/launch		POST: step/launch • orchestrate the launch of the step

abort	POST: step/abort/{step_id}
	• cancel a started step *orchestrator*:step/abort/{step_id}
			DELETE: step/abort/{step_id}
			• orchestrate rollback of *step_id* step
			  ▪ stop any datamover
			  ▪ cancel the job

status	GET: session/all • return the status of all sessions GET:session/all/{user_id} • return the status of all sessions that belong to *user_id* GET: session/{session_name} • return the status of the session *session_name*		
	GET: step/all • return the status of all steps *orchestrator*:step/status or *orchestrator*:step/status/{user_id} GET: step/status/{session_name} • return the status of all steps of the *session_name* *orchestrator*:step/status/{step_id} GET: step/status/{session_name}/{step_name} • return the status of all the *step_name* instances that belong to *session_name* *orchestrator*:step/status/{step_id} GET: step/status/{step_id} • return the status of the step with the id *step_id* *orchestrator*:step/status/{step_id}		GET: step/status/ • return the status of all steps GET: step/status/{user_id} • return the status of all steps of the *user_id* user GET: step/status/{step_id} • return the status of the step with the name *step_id*

	GET: service/all • return the status of all the services *orchestrator*:ephemeralservice/status or *orchestrator*:ephemeralservice/status/{user_id}) GET: service/{*session_name*} • return status of all services declared for *session_name* *orchestrator*:ephemeralservice/status/{service_name} GET: service/{*session_name*}/{*step_name*} • return status of all services declared for step *step_name* in *session_name* *orchestrator*:ephemeralservice/status/{service_name} GET: service/{service_name} • return status of the *session_name* service *orchestrator*:ephemeralservice/status/{service_name}		GET: ephemeralservice/status • return the status of all ephemeral services GET: ephemeralservice/status/{user_id} • return the status of all ephemeral services of the *user_id* user GET: ephemeralservice/status/{service_name} • return the status of the ephemeral service with the name *service_name*
access	GET: step/start/access/{session_name} • return as a string the salloc command to be run by the user		
show	GET: location • return the list of configured MSA location *orchestrator*:location/list GET: flavors • return information on configured flavors *orchestrator*:ephemeralservice/flavors	GET: server/resources/{server_name} • return the resources of *server_name* GET: server/resources/all • return all resources on data nodes GET: ephemeralservice/flavors • return information on configured flavors	GET: location/list • return the list of configured MSA location
			POST: step/progress • set status details for a step job (used by HSM)

2. The Orchestrator

The Orchestrator is a component of the Workflow Manager along with the Session Manager (SM) and Slurm API.

The Orchestrator provides an interface for the Session Manager allowing to execute workflow tasks on the cluster. The Orchestrator also includes the datanodes resource manager used by the Session Manager to reserve Ephemeral Services for workflows but also used by Ephemeral Services to attach to the resources reserved for starting the service itself. The Orchestrator is completely session-agnostic.

In addition to the Resource Manager (RM) endpoints (which provides an interface to manage data node resources) the Orchestrator API provides endpoint s to manage ephemeral services and workflow steps.

2.1. Resource Manager endpoints

A session is started by **reserving** resources for all ephemeral services that are described in the services section of the workflow description file. In the first implementation the Ephemeral Service (ES) are started just after reservation. In a second phase it is planned to implement a lazy policy in order to start the ES and thus use the datanode resources as late as possible (just before the service is used).

At the end of the session all ESs will be **released**. Similarly, in a second phase we can consider giving the user the possibility of stopping an ES explicitly in a dedicated step for example.

Notes:

- The Orchestrator does not know the session an ES belong to, only the SM can release the ES of a session.
- The service name used between the SM and the Orchestrator is a unique identifier composed of the user ID, the session name, and the service name. (The session name is unique for the running session therefore this service name is unique from the SM point of view).

- To provide status on ES availability the SM should pull the ES **status** from the Orchestrator. The Orchestrator API provides complete status information to the SM, which in turn passes it on to the user.

2.1.1. POST methods

The synchronous **server/allocation** method of the first version of the RM API is split into two asynchronous methods: the **ephemeralservice/reserve** POST method and the **server/allocation** GET method.

2.1.1.1. ephemeralservice/reserve

Reserve resources for an ephemeral service (used by the Session Manager).

ENDPOINT: xxx/v1.0.0/ephemeralservice/reserve (asynchronous call)

CONSUMES: at least application/json (Content-Type request header)

REQUEST BODY: v1.0.0_ephemeralservice_reserve

key	type	Optional /Required	GBF - NFS Ganesha	GBF	SBB	comment
"name"	String	required				the service name <userid>_<session name>_<service name> Note: the service name is a unique name formed by the SM from the session description information This name is provided at reservation time and used by • SM to get the reservation status (a reservation could be "FREE" if the service has not yet started or "ALLOCATED"). • slurm plugin to identify a reservation
"user"	String	required				user_id
"user_slurm_token"	String	required				the token for the execution jobs under the user identity
"type"	Enumerate	required				the service type. NFS, GBF or SBB
"servers"	Integer	optional, default 1				
"attributes"	List[key: value]	required				list of attributes needed to reserve the datanodes and start the ES
"attributes.cores"	Integer	optional	default 1		defined by Flavor	number of cores to allocate on each server. May be 0 (needed later for GBF) Exclusive with "attributes.flavor"
"attributes.msize"	Integer	optional			defined by Flavor	memory size to allocate in MB (on each server). May be 0 (needed later for GBF) Exclusive with "attributes.flavor"
"attributes.ssize"	Integer	optional			defined by Flavor	storage size to allocate in MB (on each server). Exclusive with ""attributes.gssize" and "attributes.flavor".
"attributes.gssize"	Integer	optional	required	required		Global storage size. This size will be evenly distributed among the allocated servers (to be used by GBF or other ephemeral distributed service). Exclusive with "attributes.ssize" and "attributes.flavor".
"attributes.flavor"	String	optional			required	the flavor type, as defined in the WFM configuration file. Exclusive with "attributes.cores", "attributes.msize", "attributes.ssize" and "attributes.gssize".
"attributes.targets"	String	optional			required	list of file patterns, option for SBB
"attributes.mountpoint"	String	optional	required	required		the mounting point, option for GBF
"location"	List [String]	optional				list of location from steps using the service.

RETURN CODES: v1.0.0_ephemeralservice_reserve_return

code	comment

200	reserve successfully submitted
500	reservation request failed

ACTIONS:

- search for resources and mark them as reserved :
  - if successfully then set reservation status to FREE
    (the reservation is FREE before the service is started and ALLOCATED once the service has been started)
  - fill information required by the xxx/v2.0.0/server/allocation/{service_name}
- first version:
  - create ephemeral service (~ create a persistent BB using slurm)
- second version
  - the ephemeral service will be created before starting

2.1.2. GET methods

2.1.2.1. server/allocation/{service_name}

List allocated resources for the service (used by the burst buffer slurm plugin).

ENDPOINT: xxx/v2.0.0/server/allocation/{service_name} (synchronous call)

CONSUMES: at least application/json (Content-Type request header)

PATH PARAMETERS:

key	type	comment
"service_name"	String	reservation name (<userid>_<session name>_<service names>)

RESPONSE TYPE: v2.0.0_server_allocation_response

RESPONSE BODY:

```
{
"name": String
"servers": Integer
"properties": [{
                                "server_name": String
                                "cores": Integer
                                "core_list": array[Integer]
                                "msize": Integer
                                "ssize": Integer
                          }]
}
```

v2.0.0_server_allocation_response

key	type	description
"name"	String	reservation name
"servers"	Integer	number of allocated servers
"properties"	array[v2.0.0_servers_allocation_properties]	properties of each allocation

v2.0.0_server_allocation_properties

key	type	description
"server_name"	String	allocated server name or IP adresses
"cores"	Integer	number of cores allocated on this server. Only SBB expects this to be > 0
"core_list"	array[Integer]	list of the allocated core numbers Empty list means "no binding"
"msize"	Integer	memory size (in MB) allocated on this server. Only SBB expects this to be > 0

"ssize"	Integer	storage size (in MB) allocated on this server

RETURN CODES of v2.0.0_ephemeralservice_allocation_return

code	comment
200	allocation successfully if reservation exists and status is free
500	allocation request failed because the reservation is missing
501	allocation request failed because the reservation is ALLOCATED

ACTION: set the reservation status to ALLOCATED.

Example:

```
{
"name": "user_session_service"
"servers": 1
"properties": [{
                                "server_name": "datanode1"
                                "cores": 1
                                "core_list": [0]
                                "msize": 100
                                "ssize": 200
                           }]
}
```

2.1.2.2. server/resources/{server_name}

Return available resources on the server_name server.

ENDPOINT: xxx/v1.0.0/server/resources/{server_name} (synchronous call)

CONSUMES: at least application/json (Content-Type request header)

PATH PARAMETERS:

key	type	description
"server_name"	String	data node name

RESPONSE TYPE: v1.0.0_server_resources_response

RESPONSE BODY:

```
{
        "free_cores": Integer
        "cores": Integer
        "free_core_list": array[Integer]
        "core_list": array[Integer]
        "free_msize": Integer
        "msize": Integer
        "free_ssize": Integer
        "ssize": Integer
}
```

v1.0.0_server_status_response

key	type	description
"free_cores"	Integer	number of free cores on this server.
"cores"	Integer	total number of cores on this server.
"free_core_list"	array[Integer]	list of the free core numbers Empty list means "no binding"

"core_list"	array[Integer]	list of all core numbers Empty list means "no binding"
"free_msize"	Integer	free memory size (in MB) on this server.
"msize"	Integer	total memory size (in MB) on this server.
"free_ssize"	Integer	storage size (in MB) allocated on this server
"ssize"	Integer	total storage size (in MB) on this server

RETURN CODES: v1.0.0_server_status_return

code	comment
200	status successfully returned
500	status request failed

2.1.2.3. server/resources/all

Return available resources on all servers.

ENDPOINT: xxx/v1.0.0/server/resources/all (synchronous call)

CONSUMES: at least application/json (Content-Type request header)

PATH PARAMETERS: no parameters

RESPONSE TYPE: v2.0.0_ephemeralservice_server_resources_all_response

RESPONSE BODY:

```
[
  {
        "server_name": String
        "server_resources": {
                "free_cores": Integer
                "cores": Integer
                "free_core_list": array[Integer]
                "core_list": array[Integer]
                "free_msize": Integer
                "msize": Integer
                "free_ssize": Integer
                "ssize": Integer
  }
]
```

v1.0.0_server_status_all_response

key	type	comment
"server_name"	String	
"server_resources"	v1.0.0_server_resources_response	server resources response

RETURN CODES: v1.0.0_server_resources_all_return

code	comment
200	all resources successfully returned
500	all resources request failed

2.1.3. DELETE methods

2.1.3.1. server/allocation/{service_name}

Release resources allocated for the service_name (burst buffer slurm plugin and SM).

ENDPOINT: xxx/v2.0.0/server/allocation/{service_name} (synchronous call)

CONSUMES: at least application/json (Content-Type request header)

PATH PARAMETERS:

key	type	comment
"service_name"	String	the service name used for reservation <userid>_<session name>_<service names>

Note. to remove allocation in the v2 the service name is used instead of the id used in the first version

RETURN CODES: v2.0.0_ephemeralservice_delete_allocation_return

code	comment
200	allocation successfully released
500	allocation request failed
501	allocation released failed
502	allocation not found

ACTIONS:

- delete reservation

2.2. Services and steps management endpoints

2.2.1. POST method

2.2.1.1. step/launch

Execute a step (Session Manager)

ENDPOINT: xxx/v1.0.0/step/launch (asynchronous call)

CONSUMES: at least application/json (Content-Type request header)

REQUEST BODY: v1.0.0_step/launch

key	type	Optional/Required	comment
"name"	String	required	step name <userid>_<session name>_<step name>_<index> • defined by SM
"user"	String	required	user ID
"location"	List[String]	optional	list of location
"command_env"	String	optional	environnement variables for the slurm command (key=value list)
"command"	String	required	slurm command • all variables are replaced or defined by command_env
"services"	List[v1.0.0_step_launch_service]	optional	list of services

v1.0.0_step_launch_service

key	type	Optional/Required	comment
"name"	String	required	service name
"hint"	Enumerate	optional, default RW	type of access RO, RW, WO
"datamovers"	List[v1.0.0_step_launch_service_datamover]	optional	

v1.0.0_step_launch_service_datamover

key	type	Optional/Required	comment

"name"	String	required	datamover name
"trigger"	Enumerate	required	datamover trigger : "step_start, step_stop"
"target"	Enumerate	required	data destination : NVMe, SSD, HDD, tape
"operation"	Enumerate	required	operation: copy, move, release
"elements"	List[String]	required	list of data filters

RETURN CODES: v1.0.0_step_launch_return

code	comment
200	launch step successfully submitted
500	launch step request failed

ACTIONS:

1. check or start each service from the list of services
2. execute all data movers triggered by step_start for each service, saving their jobid to be able to stop them if needed.
    - send the step_name and datamover_name to HSM API, this will be used by the HSM API to send back progress information on datamover
3. prepare the sbatch command according to the list of services
4. call slurm API
5. execute all data movers triggered by step_stop for each service, saving their jobid to be able to stop them if needed

2.2.1.2. step/progress

Set status details for a step job (HSM). For example the pourcent of a datamover progress.

ENDPOINT: xxx/v1.0.0/step/progress (synchronous call)

CONSUMES: at least application/json (Content-Type request header)

REQUEST BODY: v1.0.0_step/progress

key	type	Optional/Required	comment
"step_name"	String	required	step name <userid>_<session name>_<step name>_<index> • received from the Orchestrator when submit the job
"datamover_name"	String	required	datamover name • received from the Orchestrator when submit the job
"progress"	String	required	progress information

RETURN CODES: v1.0.0_step_launch_return

code	comment
200	step progress successfully submitted
500	step progress request failed

ACTIONS:

- update step status detail

2.2.2. GET methods

2.2.2.1. step/status/{step_id}

Return status of the step_id step. (Session Manager)

ENDPOINT: xxx/v1.0.0/step/status/{step_id} (synchronous call)

CONSUMES: at least application/json (Content-Type request header)

PATH PARAMETERS:

key	type	comment
"step_id"	String	the step name (the same used to launch the step)

RESPONSE TYPE: v1.0.0_step_status_stepid_response

RESPONSE BODY:

```
{
  "user": String
  "name": String
  "status": Enumerate
  "detail": String
  "time": String
}
```

v1.0.0_step_status_stepid_response

key	type	comment
"user"	String	user id
"name"	String	step name
"status"	Enumerate	step status, all slurm job status (BOOT_FAIL, CANCELLED, COMPLETED, CONFIGURING, COMPLETING, DEADLINE, FAILED, NODE_FAIL, OUT_OF_MEMORY, PENDING , PREEMPTED, RUNNING, RESV_DEL_HOLD, REQUEUE_FED, REQUEUE_HOLD, REQUEUED, RESIZING, REVOKED, SIGNALING, SPECIAL_EXIT, STAGE_OUT, STOPPED, SUSPENDED, TIMEOUT)
"detail"	String	status details, specify when waiting for service to start or datamovers to end. Also, information on the data transfer progress, if available.
"time"	String	time of the last status update

RETURN CODES: v1.0.0_step_status_stepid_return

code	comment
200	step status successfully returned
500	step status request failed

Example:

```
{
  "user": "string"
  "name": "string"
  "status": <BOOT_FAIL|CANCELLED|COMPLETED|CONFIGURING|COMPLETING|DEADLINE|FAILED|
                        NODE_FAIL|OUT_OF_MEMORY|PENDING|PREEMPTED|RUNNING|RESV_DEL_HOLD|REQUEUE_FED|
                REQUEUE_HOLD|REQUEUED|RESIZING|REVOKED|SIGNALING|SPECIAL_EXIT|STAGE_OUT|
            STOPPED|SUSPENDED|TIMEOUT>
  "detail": "string"
  "time": "string"
}
```

2.2.2.2. step/status

Return all step status. (Session Manager)

ENDPOINT: xxx/v1.0.0/step/status (synchronous call)

CONSUMES: at least application/json (Content-Type request header)

PATH PARAMETERS:

no parameters

RESPONSE TYPE: List[v1.0.0_step_status_stepid_response]

RESPONSE BODY:

```
[
  {
        "user": String
        "name": String
        "status": Enumerate
        "detail": String
        "time": String
  }
]
```

RETURN CODES: v1.0.0_step_status_return

code	comment
200	step status successfully returned
500	step status request failed

Example:

```
[
  {
           "user": "string"
        "name": "string"
        "status": <BOOT_FAIL|CANCELLED|COMPLETED|CONFIGURING|COMPLETING|DEADLINE|FAILED|
                   NODE_FAIL|OUT_OF_MEMORY|PENDING|PREEMPTED|RUNNING|RESV_DEL_HOLD|REQUEUE_FED|
              REQUEUE_HOLD|REQUEUED|RESIZING|REVOKED|SIGNALING|SPECIAL_EXIT|STAGE_OUT|
          STOPPED|SUSPENDED|TIMEOUT>
         "detail": "string"
        "time": "string"
    }
]
```

2.2.2.3. step/status/{user_id}

Return status of all steps for a the user_id. (Session Manager)

ENDPOINT: xxx/v1.0.0/step/status/{user_id} (synchronous call)

CONSUMES: at least application/json (Content-Type request header)

PATH PARAMETERS:

key	type	comment
"user_id"	String	the user id

RESPONSE TYPE: List[v1.0.0_step_status_stepid_response]

RESPONSE BODY:

```
[
  {
        "user": String
        "name": String
        "status": Enumerate
         "detail": String
        "time": String
  }
]
```

RETURN CODES: v1.0.0_step_status_userid_return

code	comment
200	status successfully returned
500	status request failed

Example:

```
[
  {
         "user": "string"
        "name": "string"
        "status": <BOOT_FAIL|CANCELLED|COMPLETED|CONFIGURING|COMPLETING|DEADLINE|FAILED|
                   NODE_FAIL|OUT_OF_MEMORY|PENDING|PREEMPTED|RUNNING|RESV_DEL_HOLD|REQUEUE_FED|
              REQUEUE_HOLD|REQUEUED|RESIZING|REVOKED|SIGNALING|SPECIAL_EXIT|STAGE_OUT|
          STOPPED|SUSPENDED|TIMEOUT>
         "detail": "string"
        "time": "string"
   }
]
```

2.2.2.4. ephemeralservice/status/{ephemeralservice_id}

Return status of the ephemeralservice_id service. (Session Manager)

ENDPOINT: xxx/v1.0.0/ephemeralservice/status/{ephemeralservice_id} (synchronous call)

CONSUMES: at least application/json (Content-Type request header)

PATH PARAMETERS:

key	type	comment
"ephemeralservice_id "	String	the service name (the same used to reserve the service)

RESPONSE TYPE: v1.0.0_ephemeralservice_status_ephemeralserviceid _response

RESPONSE BODY:

```
{
  "user": String
  "name": String
  "status": Enumerate
  "detail": String
  "time": String
}
```

v1.0.0_ephemeralservice_status_ephemeralserviceid_response

key	type	comment
"user"	String	user id
"name"	String	ephemeral service name
"status"	Enumerate	service status: STAGINGIN, STAGEDIN, ALLOCATED, STAGINGOUT, STAGEDOUT, STOPPED, TEARDOWN
"detail"	String	status details
"time"	String	time of the last status update

RETURN CODES: v1.0.0_ephemeralservice_status_ephemeralserviceid_return

code	comment

200	ephemeral service status successfully returned
500	ephemeral service status request failed

Example:

```
{
  "user": "string"
  "name": "string"
  "status": <STAGINGIN|STAGEDIN|ALLOCATED|STAGINGOUT|STAGEDOUT|STOPPED|TEARDOWN>
  "detail": "string"
  "time": "string"
}
```

2.2.2.5. ephemeralservice/status

Return status of all ephemeral services. (Session Manager)

ENDPOINT: xxx/v1.0.0/ephemeralservice/status (synchronous call)

CONSUMES: at least application/json (Content-Type request header)

PATH PARAMETERS:

no parameters

RESPONSE TYPE: List[v1.0.0_ephemeralservice_status_ephemeralserviceid _response]

RESPONSE BODY:

```
[
  {
    "user": String
    "name": String
    "status": Enumerate
    "detail": String
    "time": String
 }
]
```

RETURN CODES: v1.0.0_ephemeralservice_status_return

code	comment
200	ephemeral service status successfully returned
500	ephemeral service status request failed

Example:

```
[
  {
    "user": "string"
    "name": "string"
    "status": <STAGINGIN|STAGEDIN|ALLOCATED|STAGINGOUT|STAGEDOUT|STOPPED|TEARDOWN>
    "detail": "string"
    "time": "string"
  }
]
```

2.2.2.6. ephemeralservice/status/{user_id}

Return status of all ephemeral services of the user_id. (Session Manager)

ENDPOINT: xxx/v1.0.0/ephemeralservice/status/{user_id} (synchronous call)

CONSUMES: at least application/json (Content-Type request header)

PATH PARAMETERS:

key	type	comment
"user_id"	String	the user id

RESPONSE TYPE: List[v1.0.0_ephemeralservice_status_ephemeralserviceid _response]

RESPONSE BODY:

```
[
  {
    "user": String
    "name": String
    "status": Enumerate
    "detail": String
    "time": String
 }
]
```

RETURN CODES: v1.0.0_ephemeralservice_status_userid_return

code	comment
200	ephemeral service status successfully returned
500	ephemeral service status request failed

2.2.2.7. location/list

Return the list of location (used by Session Manager to paresse the WFD file)

ENDPOINT: xxx/v1.0.0/location/list (synchronous call)

CONSUMES: at least application/json (Content-Type request header)

PATH PARAMETERS:

```
No parameters
```

RESPONSE TYPE: v1.0.0_location_list_reponse

list of v1.0.0_location

v1.0.0_location

key	type	description
"name"	String	name of location

RESPONSE BODY:

```
[
  {
    "name": String
  }
]
```

RETURN CODES: v1.0.0_location_list_return

code	comment
200	location list successfully returned
500	location list request failed

2.2.2.8. ephemeralservice/flavors

Return the list of flavor (used by Session Manager to paresse the WFD file)

ENDPOINT: xxx/v1.0.0/ephemeralservice/flavors (synchronous call)

CONSUMES: at least application/json (Content-Type request header)

PATH PARAMETERS:

```
No parameters
```

RESPONSE TYPE: v1.0.0_ephemeralservice_flavors_response

list of v1.0.0_ephemeralservice_flavor

v1.0.0_ephemeralservice_flavor

key	type	description
"name"	String	flavor name
"cores"	Integer	number of cores, expects this to be > 0
"msize"	Integer	memory size (in MB) , expects this to be > 0
"ssize"	Integer	storage size (in MB), optional

RESPONSE BODY:

```
[
  {
    "name": String
    "cores": Integer
    "msize": Integer
    "ssize": Integer
  }
]
```

RETURN CODES: v1.0.0_ephemeralservice_flavors_return

code	comment
200	flavor list successfully returned
500	flavor list request failed

2.2.3. DELETE methods

2.2.3.1. ephemeralservice/release/{service_name}

Stop the service name (session manager).

ENDPOINT: xxx/v1.0.0/ephemeralservice/release/{service_name} (asynchronous call)

CONSUMES: at least application/json (Content-Type request header)

PATH PARAMETERS:

key	type	comment
"service_name"	String	service name (<userid>_<session name>_<service name>)

RESPONSES: v1.0.0_ephemeralservice_release_return

code	comment
200	service successfully released

500	released request failed not found or not allocated

ACTIONS:

- destroy the ephemeral service (~ destroy the persistent BB using slurm)

2.2.3.2. step/abort/{step_id}

Cancel a step. (Session Manager)

ENDPOINT: xxx/v1.0.0/step/abort/{step_id} (asynchronous call)

CONSUMES: at least application/json (Content-Type request header)

PATH PARAMETERS:

key	type	comment
"step_id"	String	the step name (the same used to launch the step)

RESPONSES: v1.0.0_step_abort_stepid_return

code	comment
200	abort request successfully returned
500	abort request failed

ACTIONS:

- stop any datamover
- cancel job